

REMARKS

Claims 1, 3-16, 18-30 and 32-42 are pending in the present application and stand rejected under 35 U.S.C. § 103. Applicants would like to direct Examiner to a significant misinterpretation of the prior art prior to proceeding with an appeal brief.

The Examiner has consistently contended that counters of the PMU 90 in the Krishnaswamy reference teach or suggest the “memory array,” as claimed by Applicants. Applicants respectfully submit herewith objective evidence that such an argument is untenable and incorrect to those skilled in the art even its broadest reasonable interpretation.

A “memory” (and “memory array”) as used in the general terminology of computer architecture is distinguishable from a “counter.” For example, Professor Doug Burger of the University of Texas at Austin teaches a basic computer architecture course for undergraduate computer science students. In slides 2 and 25 (attached herewith) of lecture 2 in Professor Burger’s class (http://www.cs.utexas.edu/users/dburger/teaching/cs352-s05/lectures/Lecture_2.pdf), the basic concepts of a memory are defined. Examples of memory include SRAM and DRAM. For another example, an exemplary SRAM array diagram from an undergraduate digital integrated circuits class at the University of California at Berkeley is also attached herewith (http://bwrc.eecs.berkeley.edu/Classes/ICDesign/EE141_f04/Project/Project2004.pdf).

On the other hand, as shown in slide 17 (attached herewith) of Professor Burger’s lecture, a “counter” is built with latches or flip flops, which are “clocked elements,” as differentiated from the “memory” slides 2 and 25. An exemplary binary counter consisting of flip-flops is also attached herewith (<http://hyperphysics.phy-astr.gsu.edu/hbase/electronic/bincount.html>).

BEST AVAILABLE COPY

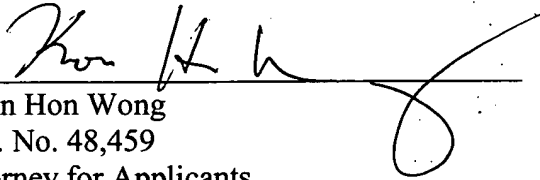
It should further be noted that while memory arrays are addressed, counters cannot be "addressed" as claimed in claim 33. Attached herewith is a paper detailing memory array addressing. The paper, co-authored by Konrad Lai, director of Intel's Microprocessor Research Lab in Oregon, discusses tradeoffs in memory array design. Please note that this paper also references a work of Professor Burger's.

Applicants respectfully request reconsideration of the claim rejections in view of the objective findings submitted above. In particular, Applicants hope to have clarified any misinterpretations of the science. Withdrawal of the claim rejections under 35 U.S.C. §103(a) is respectfully requested.

In view of the foregoing remarks, it is respectfully submitted that all the claims now pending in the application are in condition for allowance. Early and favorable reconsideration is respectfully requested.

Respectfully submitted,

By:


Koon Hon Wong
Reg. No. 48,459
Attorney for Applicants

F. CHAU & ASSOCIATES, LLC
130 Woodbury Road
Woodbury, New York 11797
Telephone: (516) 692-8888
Facsimile: (516) 692-8889

Dynamic Addressing Memory Arrays with Physical Locality

Steven Hsu[§], Shih-Lien Lu[§], Shih-Chang Lai^{*}, Ram Krishnamurthy[§], Konrad Lai[§]

[§]Microprocessor Research, Intel Labs
Intel Corporation
steven.k.hsu@intel.com

^{*}Dept. of Electrical & Computer Engineering
Oregon State University

Abstract

As pipeline width and depth grow to improve performance, memory arrays in microprocessors are growing in entries and ports. Arrays will increase in physical size, which prolongs the access time due to wiring delay. In order to boost clock frequency, these memory arrays must take multiple cycles to complete an access. This delays the scheduling of dependent instructions and affects overall performance. This paper proposes a different circuit organization to enable fast and slow accesses solely dependent on physical locality. Since the access time depends on a fixed physical location, it is pre-determined to scheduling dependent instructions. Furthermore, this paper presents a mechanism to re-configure the address decoding of the physical register file to increase the occurrence of fast accesses. Detailed circuit simulation using this proposed method determines the access cycle time. Reduction in average access cycle time for the register file and the first level data cache recovers 73% of the IPC degradation.

1. Introduction

One of the concerns in continuing the improvement of superscalar microprocessor performance is scalability. As processors become wider to exploit instruction level parallelism (ILP), the processor's physical sizes also increase. Larger physical size impedes the growth of processor clock frequency. Even though transistors are becoming faster as technology scales, wiring delays are not becoming shorter [1][2]. Since interconnect delay is not scaling as fast as transistor delay, longer wiring delays will impact overall performance growth, as it limits the performance of memory arrays used throughout modern microprocessors [3]. Processors are becoming deeper by employing finer and finer pipeline stages [4][5][6], in order to achieve the optimal balance between frequency and ILP for peak performance.

One of the important memory arrays in a superscalar processor is the register file. It is used to communicate results between computations. To remove anti-dependencies, architecture registers are renamed to physical registers. The physical register file is usually much larger than the architecture register file in order to exploit parallelism. Moreover, since there are multiple instructions issued in a cycle, physical register files need to be multi-ported, resulting in the rapid size growth of register files. In order to shorten the access time, a register file is designed with single-ended full-swing dynamic circuits instead of the differential small swing design used in a large cache structure. These factors all contribute to the size growth of the register file and thus increase the access time. For example, a typical 256 entries 64-bit 8-read / 4-write register file will need more than half a million transistors to implement the array itself, excluding the address decoding, peripheral, and control circuits.

The sheer size of the register file will adversely impact the access time. Currently, a single cycle register file latency access will no longer be feasible because of the amount of interconnect the signal needs to travel from address generation to the output of the memory array. As cycle times decrease, the amount of interconnect distance a signal can travel in a single cycle will decrease. Thus, designs are forced to allocate multiple cycles for a single access to the register file, which in turn affects the performance of the microprocessor [7].

Two-level register file architectures have been proposed [8][9] to mitigate the latency problem of multi-cycle register files. This type of design employs a smaller but faster memory structure containing a part of all information from the original register file. Most of the accesses will be to the register file cache, which is faster. It improves the average access latency to the register file by exploiting locality. Inclusion is maintained, that is, data in the cache should be found in the register file. With inclusion, tag checking must be performed to find out if the data is in the cache or not. This introduces complexity in scheduling of dependent instructions

because the presence of accessed data in the cache determines the total latency of the instruction. Also, this scheme only introduces another level of hierarchy in the memory pyramid.

Thus it is advantageous to have a register file design that has shorter average latency but has known access time before an instruction is scheduled. This paper proposes a circuit and organization approach to the traditional register file design, which exploits physical locality and has a known access time at the time of mapping reconfiguration to increase the frequency of usage in the fast array of the register file. This method checks the physical register file, which is organized as a circular queue at the decode time. Whenever a slower entry is allocated for storing the destination, it checks the pointers used to keep track of the circular queue. If the faster counter part of the queue is un-used, a bit is set to indicate that the data can be duplicated into that entry. Upon completing the execution of the instruction, the result is stored in both the original place and the duplicated place if the bit is set. Subsequent read from the register file's address logic is designed so that it will read from the faster part of the array. This method only requires a small 1-bit wide table to indicate an entry has a valid duplicated copy or not. Some minor modification to the address decoding logic is also needed.

This paper is organized as follows. First, this paper will discuss the base line micro-architecture that this proposed scheme applies. This paper also explains the newly proposed address re-mapping scheme used to improve average access latency in the same section. In the next section, the paper will then discuss the best-known design of register file circuits and the modifications to enable variable latency. In Section 4 the simulation results from circuit simulation and microarchitecture simulation is presented. Lastly, this paper will draw conclusions in the final section.

2. Register File Architecture

2.1 Conventional Register File

Current high-performance processors all utilize register renaming to remove anti-dependencies. One implementation scheme utilizes three array structures in addition to the architecture register file. The array structures include: the rename table, the physical register file (RF) and the re-order buffer (ROB). The rename table contains the physical register address where the logical addresses are mapped. The physical register contains the actual data value before it is committed. The ROB acts as the instruction window and contains instruction operation code, source and renamed source scheduling. Part of the register file is faster than the other half because it is closer to the peripheral circuit. The

access time depends solely on the location of the register's physical location and thus can be pre-determined prior to the scheduling of instructions.

If the access time to the queue depends on the physical location, then it is desirable to increase the occurrence of accesses to the fast array. This paper further proposes a simple address decoding modification, allowing address-destination bindings (physical) and status. When an instruction is decoded, its destination register is assigned to a free physical register (allocation). Source registers also lookup this renaming table to obtain the physical register binding for later read access. This instruction allocates an entry in the ROB and places the physical register bindings in it. Later, instructions in the ROB are waken up and scheduled for execution. Before the instructions are executed they pass through the physical register file to acquire their data.

A physical register is released back into the free list when the next instruction that writes to the same architecture register (which has been renamed to a different register) commits its result. There are several proposed methods to release a physical register earlier. However, these methods are orthogonal to what is proposed in this paper. To simplify the discussion, we treat the physical register file as a circular queue that can be accessed randomly. There are two types of applicable operations. First, there are the queue management operations. As instructions pass through the decode stage, entries in the queue are allocated by incrementing a tail pointer. As physical registers are released at the commit stage, a head pointer is incremented. Second, there are regular random read and write accesses. Read access occurs during the register read (RE) pipeline stage and write access occurs during the write back (WB) stage.

Borch et al. points out the impact of micro-architecture loops on performance in the recent paper titled "Loose Loops Sink Chips" [7]. In the same paper, the authors also show that latency from the instruction queue issue (IQ) stage to the execution (EX) stage is more critical than the latency from the decode stage (DEC) to IQ. Accessing the physical register file (RE) occurs during this more critical loop. Therefore, it is advantageous to reduce the read access time of the physical register file. To improve this instruction loop performance, this paper proposes to design the register file to take advantage of the physical locality. In a memory array there are entries that are located closer to the peripheral circuit and thus can be accessed faster than entries that are physically located further away.

2.2 Variable Cycle Register File

In order to achieve fast access time with multiple ports, a register file's data lines (called *bit-lines*) are designed with single-ended full swing dynamic circuits. Each

storage cell, a cross-coupled inverter, drives a pull-down n-type device, which is connected in series to the dynamic node (bit-line). With leakage current increasing, the number of rows (entries) that are on bit-line segment is limited in order to read out data reliably. For the sub-100nm generation technology, this number is less than ten. This bit-line is referred to as the local bit-line (LBL). Typically two local bit-lines are merged using a static NAND gate and the output of the NAND gate is fed into another dynamic NOR gate with other merged local bit-lines. Depending on the number of register entries in the register file there may be several levels of these dynamic-static (domino) stages arranged as a tree structure implementing a wide OR operation. For example, a 256-entries register file is arranged with 8 entries per local bit-line. Two local bits are merged with a static NAND gate. Eight of these merged local bit-lines (a total of 16 local bit-lines each) are combined on a global bit-line. Thus, each global bit-line has 128 entries. Finally, two global bit-lines are merged with a static NAND to complete 256 entries.

Usually, this tree structure is balanced perfectly to allow a uniform access time independent of which entry is read. In our previous example, the final merged NAND usually is placed in the middle of the two global bit lines, and a final wire segment covering the length of half the entries must be used to bring the final output out of the array. It is not essential to make the tree well balanced having the same access time as long as the scheduler knows about it ahead of time. Thus, the organization can be arranged to allow the entries physically closer to the peripheral circuit that drive the output to be read out and latched first. The remaining half can be read out through the original wire used to read out the merged output. In this way the access to the first half is faster than the original speed while the access to the second half remains the same. In Section 3, this paper will discuss the detail arrangement and quantify the speed difference using a next generation technology. The proposed technique observes that the register file may be designed to enable faster access to the front half of the entries than access over the end half due to physical locality.

However, the locations of the fast access are fixed since this is dependent on the physical layout of the actual transistors. In the example, at least half of the time the access time, is the same as before. Maximizing performance can be achieved by increasing the probability that an access will hit the faster part of the register file. A simple method is proposed in section 2.3.

2.3 Increasing the Use of Fast Registers

To summarize, the proposed method duplicates new data into an unused space whenever possible. When the space storing the duplicated data is needed, it is simply

written over. A small 1-bit table with $n/2$ entries is maintained to indicate that an entry with slower access time has a valid duplicated copy, where n is the number of entries in the register file. This table is called the *swap register table*. Any access to a slow entry with a valid duplicated copy is automatically decoded to load from the duplicated site, which has shorter access latency. Whenever the corresponding bit is set, there is a valid duplicated entry.

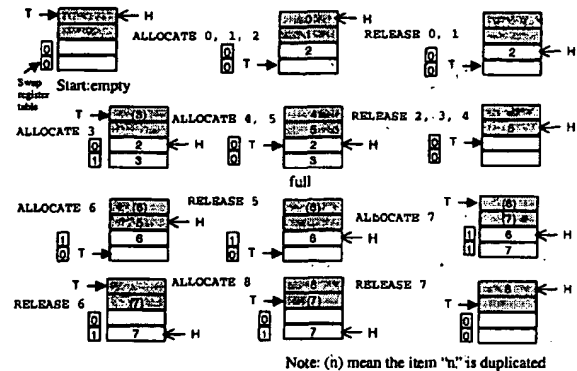


Figure 1. Re-configurable Addressing Example

The proposed method can be explained by using a simple circular queue with four entries as an example in Figure 1. Half of the entries (the shaded top two entries) have a shorter access time. The remaining half of entries (un-shaded bottom two entries) has a longer access time. The data items are numbered to be loaded into the queue from 0 on. Figure 1 details an example of a sequence of operations. This circular queue is maintained with two pointers HEAD (H) and TAIL (T). Initially both the HEAD (H) and TAIL (T) pointers are pointing to the first location of the array. At this time, the queue is now empty. The first set of operations consist of allocating three entries (0, 1, 2) for writes. The TAIL pointer is moved to point to the fourth entry while the HEAD pointer remains at the beginning as shown. If there are random read accesses to the queue at this time, then two-thirds of the accesses will hit the portion of the queue with shorter access time. The second set of operations consists of releasing two items from the queue. Afterwards, the HEAD pointer points to entry 3 and the TAIL pointer remains at the same location. Accesses to the queue at this time will have a longer access time. Later, a new allocation for write occurs. Since the "corresponding" entry in the faster half is un-occupied, a duplicate copy is written. A fast entry can be determined to be un-occupied by comparing the value of the HEAD pointer and the entry address. If duplication is allowed, data is written into both places. This is achieved through a modification to the decoding logic to allow two word lines to be enabled during the write operation. A bit will be set to

indicate that a valid duplicated version is available. After this bit has been set, accesses to the queue will land on the faster part half of the time through a modified decoding logic. If the entry has not been duplicated, then all accesses will hit the slower part instead. This example continues with two more allocations (entries 0, 1) for writing data 4 and 5. Then the queue is full and half of the accesses will have shorter cycle time. Notice that the first new allocation basically writes over the duplicated item. The bit used to indicate the duplicated data in the faster section is cleared when the entry is over written. A few more operations continue in the example as shown.

It is worthwhile to note that a register file entry is allocated, but not written into immediately. There is a time gap between allocation and actual data storage. It may appear that the register swap table can be set and reset not at the allocation and release time as described above, but at the time when data is actually written giving even more opportunities to place the data in the faster section. However, this would result in the loss of the ability to pre-determine access time at the issues time. The following section will detail the circuit design of the proposed variable latency register file.

3. Register File Circuit Design

Circuits used to build memory arrays in high performance microprocessors vary from small-signal caches to large-signal register files. The small-signal design utilizes two bit-lines per data bit since it requires differential reads. A small voltage swing difference is developed during the read process. This small differential voltage is amplified and outputted. The large-signal design needs only one bit-line per data bit (per port). This bit-line is usually pre-charged, and the read process either discharges the bit-line or leaves it unchanged. Small-signal design is more suited for single-ported dense memory arrays. Dense memory arrays are slower because of the increased density. Large-signal design is more suitable for arrays with a large number of ports, since it can meet stability requirements. It is denser to use single-ended large signal design if the number of ports are larger. This paper will focus on large-signal register files and first-level cache. In the near future, even the first-level cache will be implemented with large-signal circuits to reduce the access time due to increasing leakage currents. These multi-ported register files are implemented in large signal 2-phase single-rail domino circuit [10]. 2-phase domino employs simple 50% duty cycle clocks and completely hides the pre-charge time in dynamic logic, allowing only useful read evaluation work to be part of the critical path.

3.1 Conventional Register Files

The number of read and write ports in a register file are dictated by the issue width. For example, a 4-issue superscalar processor will have 8 read ports and 4 write ports. Since each port will need a bit-line, there will be total of 12 bit-lines and 12 word lines.

3.1.1. Circuit Organization. Figure 2 shows an example design of a conventional register file for a 256 x 64-bit 8 read / 4 write ports register file with the physical dimensions depicted. The register file decoder and array occupies a dense layout of 865 μ m x 100 μ m. This dimension is obtained from actual layout using an advanced CMOS 100nm process. This register file is pipelined and requires three-cycles for a read access. As shown, the array is split into half with the decoder in the middle to reduce word line length and achieve higher performance. The first cycle is attributed to the decoding of the address. Typically, the decoder is non-critical and can easily meet timing. The second cycle is attributed to word line activation, local bit line discharge and a section of the global bit line discharge. The global bit line is partitioned into separate bit lines with domino repeaters due to the excessive resistive interconnect length. The third cycle is attributed to the final global bit line merge between arrays. The final global merge forces the design to a 3-cycle register file, since delay is heavily dominated by interconnect. The data is delivered at the bottom of the middle array for a balanced access time in the receiving functional unit.

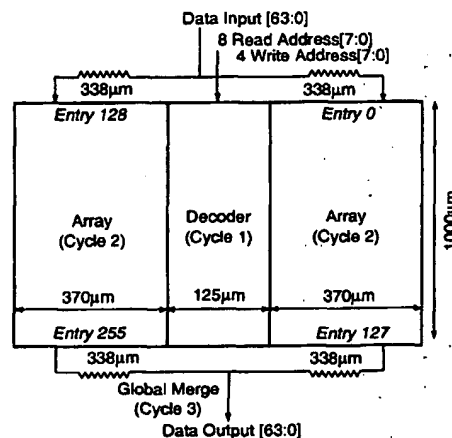


Figure 2. Example Conventional Register File Design

3.1.2. Address Decoder. Decoders are implemented as wide AND gates. A 256-entry register file needs 8 address inputs to determine which entry is selected. There must be a decoder for each write and read port. Dynamic or static CMOS logic implementations are both

acceptable; static CMOS logic design gives lower power since clock nodes are not discharged and charged every cycle and maintains a low activity factor. The design implements a fully static decoder, since it can be done comfortably in a single clock cycle. The decoder feeds the word line drivers and is set up well ahead of the rising edge of the second cycle.

3.1.3. Memory Cell. Figure 3 shows an example 8 read 4 write register file cell. Heavily multi-ported arrays are interconnect dominated by minimum pitch metal 3 (M3) tracks for word lines and minimum pitch metal 2 (M2) or metal 4 (M4) tracks for bit-lines. Typically, the cell area is dictated in the x-dimension by M3 interconnects. The width (x direction) is the M3 pitch multiplied by the # of read + write ports. The M2 and M4 wires dictate the y-dimension. Thus the height (y direction) equals the M4 wire pitch multiplied by the # of read + write ports + power. As the number of ports increases, the cell area will increase also since it is interconnect limited.

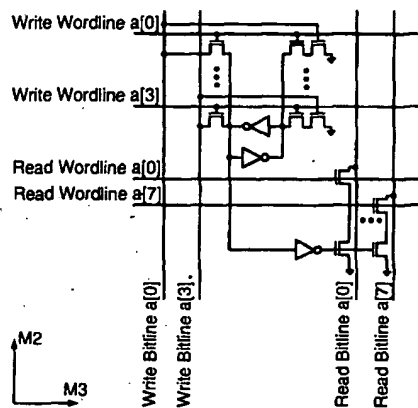


Figure 3. Storage Cell of an 8-Read 4-Write Register

3.1.4. Local and Global Bit-lines. Local bit-lines are implemented in single rail 2-phase domino logic. Domino logic is the most efficient way to implement large fan-in logic gates and will give the highest performance. The bit-lines are limited by leakage and interconnect length. The leakage of the read ports on the local bit-line injects noise on the dynamic node during evaluation. The dynamic node is only weakly held by a keeper to prevent the leakage from causing a false evaluation. The resistance of the interconnect sets the maximum length allowed for a global bit-line evaluation in a single phase. The local bit-line is routed on minimum pitch M2, and the global bit-line (GBL) is routed on a higher-level metal minimum pitch M4. In this example design (Figure 3) the half-shielded local bit-line is 61 μ m in length, while global level bit-lines are 460 μ m long. A

static 2-input NAND gate merges each dynamic local bit-line. The NAND gate feeds into a dynamic global bit-line, which is again a wide domino OR gate. Figure 4 illustrates the bit-line circuits.

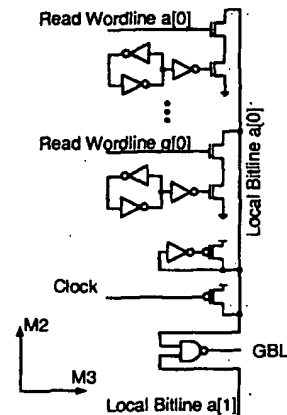


Figure 4. Bit-line Structure

3.2 Register File with Physical Locality

3.2.1. Architecture. A register file can be implemented to exploit physical locality. Using a simple rearrangement of the final global merge of the register file, the array can now deliver an entry in two or three cycles, depending on the physical location of the entry. Worst case read access cycle time in this new register file is the same as the worst case read accesses cycle time for the conventional register file. Therefore, worst case performance can be no worse than conventional performance. However, best case read access latency would be 1 cycle faster than the conventional case.

3.2.2. Physical Locality. Physical locality uses the physical layout position of the entry to allow for two or three cycle register file accesses. Figure 5 shows an example design of the newly proposed register file with physical locality. Since entries 0 to 127 are physically closer, processors should use these entries more, over entries 128 to 255. The final 676 μ m global merge is now designed to complete the critical path of entries 128 to 255. Entries 0 to 127 are all two-cycle accesses because of physical locality, whereas entries 128 to 255 are all three-cycle accesses. The final merge is all clubbed into the third cycle for entries 128 to 255, while it does not affect the fast array for entries 0 to 127.

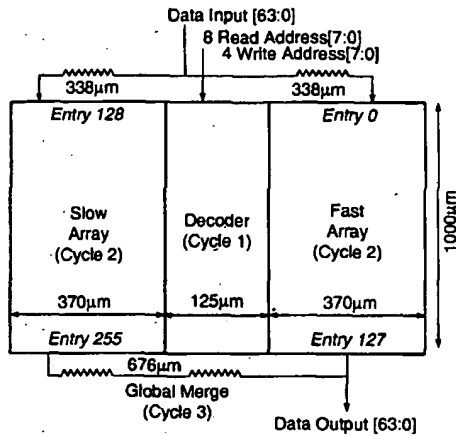


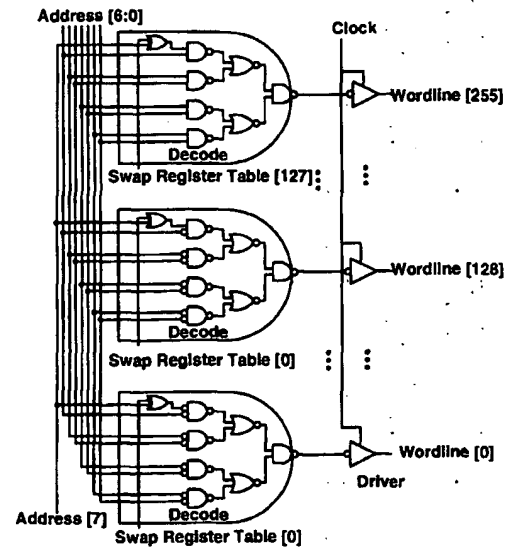
Figure 5. Register File With Physical Locality

3.3 Dynamic Addressing Decoder

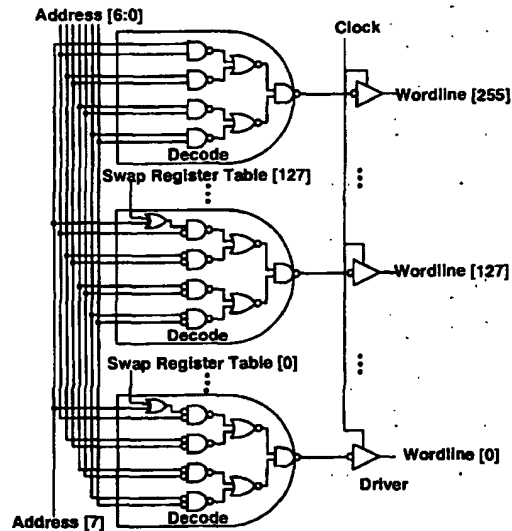
Address decoding converts an encoded address into one-hot word line signals, which selects the entry for read access. If a valid duplicated copy is available, indicated by the corresponding swap register bit being set, the most significant bit (MSB) of the address is inverted for read access. This inversion can be achieved through a simple XOR gate as indicated in Figure 6(a). One of the inputs to the XOR gate is the MSB of the address. The other input is from the swap register table. Each entry of the swap register table drives two XOR gates – the i^{th} and the $(i+(n/2))^{\text{th}}$. In Figure 6, the first entry of the swap register table (denoted as entry [0]) drives the first and the 128th, for example. If the incoming address is 128 (100000002) and the swap register table [0] is set then wordline[0] will be selected (high) while wordline[128] will not be selected (low). Using this method, the processor is able to take advantage of the faster portion of the register file as much as possible. When the data is ready to be written into a slow entry and there is an un-occupied entry in the fast array, indicated by the bit set in the swap register, then both word lines are enabled. This is accomplished through XOR gates again as shown in Figure 6(b).

4. Simulation and Result Discussion

In order to study the effect of the proposed method, circuit-level simulations as well as microarchitecture simulation are both performed. Circuit simulation provides a comparison of the detail delay information between the conventional design and our proposed design with interconnects delay included. Microarchitecture simulation studies the impact of having variable latency on register file and data cache read.



(a) Read Port Address Decoder



(b) Write Port Address Decoder

Figure 6. Address Decoder Circuits for Dynamic Variable Access Time

4.1 Circuit Level Simulation

A 4 GHz 100nm 256 x 64b 8 read / 4 write register file was designed and simulated with interconnect and parasitic loads. Circuit simulation was done using a speculative 100nm process at 1.2V and 110° Celsius using all high-threshold devices. It was found that the conventional register file design was forced to include an extra cycle in the read array path for the final global merge. Upon altering the configuration, the register file

with physical locality was able to achieve a fast and slow access within the 4 GHz cycle time. Figure 7 illustrates the delay-timing break down of the circuit simulation for the most critical part of the read operation – the array read. For the conventional read access, the first phase of the second cycle includes the delay of the word line (WL) activation and local bit-line (LBL) evaluation. The second phase of the second cycle includes the two-level global bit-line (GBL) merge. Lastly, the final global merge (FGM) completes in the third cycle and merges the two arrays with a dynamic to static (S) NAND interface. The fast access completes within a single cycle of 250ps and is not impacted by a final global merge (FGM), but still has the dynamic to static (S) NAND interface. The slow access, while very similar to the conventional case, uses the first phase of the third cycle to evaluate the final global merge (FGM) and dynamic to static (S) interface. The unlabeled white space in Figure 7 includes clock skew, jitter, and margin.

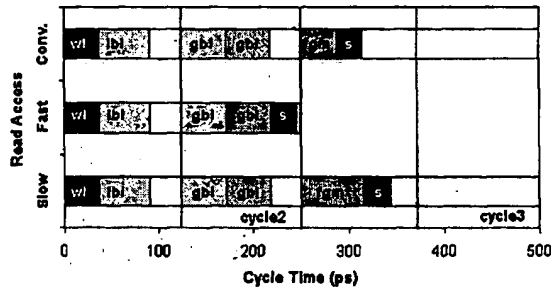


Figure 7. Data Read Delay Timing

4.2 Architecture Simulation

The architectural simulation used a modified version of the SimpleScalar toolset [11] to evaluate the performance impact on varying the register read cycle. Ten integer SPEC 2000 benchmarks [12] are used to evaluate the effect. After skipping the first 500 million instructions, statistics for 500 million committed instructions are collected.

4.2.1. Simulation Configuration. The base model processor is a seven-stage pipeline micro-architecture: Fetch (FE), Decode/Rename (DE), Schedule (SC), Register read (RE), Execution (EX), Write back (WB) and Commit (CM). The front-end of the processor has less impact on the IPC as suggested by [7]. Thus, the FE and DE take only one cycle. The schedule stage has a 2-cycle latency, while the register read stage has a 3-cycle latency. Execution cycle latency depends on the type of operation as well as computing resource. For example, our base line model uses a 6-cycle first-level data cache.

This number is suggested by [5] and through our circuit level estimation. All other stages are single-cycle. Furthermore, we scheduled the instruction a data-ready cycle to eliminate the effect of data speculation. Table 1 summarizes the detail processor configuration.

Table 1. SimpleScalar Configuration

Description	Configuration
Fetch, Decode, Schedule, Commit Width	4
Fetch/Decode/Schedule/Register/Execute/Writeback/Commit stage latency	1/1/2/3/FU-latency/1/1
Architecture Register File IO port	8 reads/ 4 writes
Branch Predictor	16-bits Gshare
Branch Prediction Table	8K-entry, 8 way
ROB/LSQ size	256/128
L1 I/D cache	32KB/32KB, 4-way, 32B line size
L1 I/D cache hit latency	1/6 cycles
L2 cache	4MB
L2 / memory latency	10/100 cycles
# of pipelined integer ALU/MULT/DIV	4/1/1
Integer ALU/MULT/DIV	2/6/40
Floating-point Adder/MULT/DIV	4/8/48
Cache Read/Write port	2

This base model, which has a static RF access time of three cycles (RF-S-3) and static level-1 data cache latency of 6 (C-S-6), represents our performance (IPC) lower bound. The IPC performance upper bound is a processor with 2-cycle register read stage (RF-S-2) and 5-cycle L1 D-cache latency (C-S-5). With no address decoding reconfiguration, our example 256x64 register file circuit design allows half of the 256 entries (0 to 127) to have faster access time (2-cycle). The other half (registers 128 to 255) of the register entries adds an extra cycle to its access time. Therefore, when an instruction is decoded a new physical register is allocated. If this entry is among the first half of the register file (from 0th to 127th), then dependent instructions are scheduled to wakeup with 2-cycle RE latency. Otherwise, this instruction allocates a physical register among the slower portion and all dependent instructions will be scheduled according to a 3-cycle RE latency.

Table 2. Simulation Test Cases

	Type	Shorthand	Description
Register File	Static 2 cycle	RF-S-2	Conventional 2 cycle reads
	Static 3 cycle	RF-S-3	Conventional 3 cycle reads
	Static 2/3 cycle	RF-S-2_3	Static 2 or 3 cycle reads
	Dynamic 2/3 cycle	RF-D-2_3	Dynamic Addressing 2 or 3 cycle reads
Data Cache	Static 5 cycle	C-S-5	Conventional 5 cycle reads
	Static 6 cycle	C-S-6	Conventional 6 cycle reads
	Static 5 / 6 cycle	C-S-5_6	Static 5 or 6 cycle reads

This design is called the RF-S-2_3 model. The implementation of the dynamic address decoding design is called the RF-D-2_3 model. Table 2 summarizes the notation used in reporting results in the following sections.

4.2.2. Experimental Results. As pipeline stages are partitioned into multiple cycles to keep the clock frequency up, IPC is degraded. For example, one extra RE cycle (3 vs. 2) and one extra level 1 data cache latency

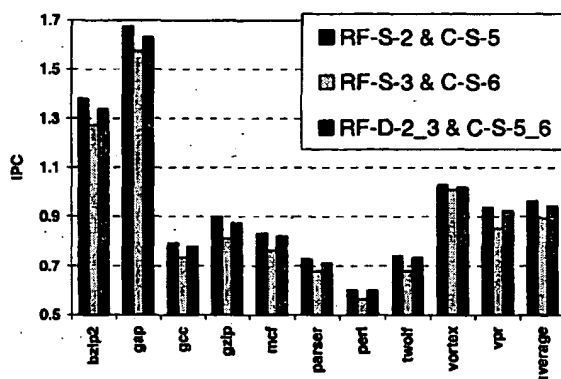


Figure 8. IPC for Upper bound (RF-S-2 & C-S-5), Lower bound (RF-S-3 & C-S-6), and Proposed (RF-D-2_3 & C-S-5_6)

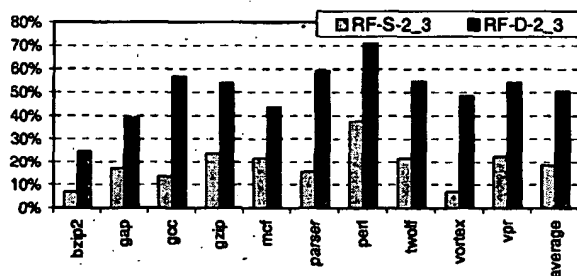


Figure 9. IPC Recovered for Register File Static and Dynamic 2/3 Cycle Reads

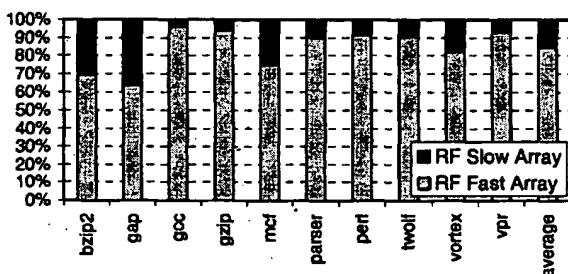


Figure 10. Distribution of Committed Instruction hitting the Register File for RF-D-2_3

(6 vs. 5) degrades the IPC by more than 7%. It is the goal of this proposed approach is to recover the IPC loss while maintaining the high clock frequency. Figure 8 shows the IPC upper bound, lower bound, and best-proposed case. The upper bound is the 2-cycle register file read with a cache latency of 5 cycles (RF-S-2 & C-S-5). The lower bound is the 3-cycle register file read with a cache latency of 6 cycles (RF-S-3 & C-S-6). The best proposed scheme uses the dynamic addressing register file and variable latency cache (RF-D-2_3 & C-S-5_6). As shown, the proposed scheme achieves very close to the actually IPC of the upper bound, recovering much of the IPC lost. The following sections will detail the various schemes.

4.2.3. Instruction Per Cycle (IPC) Recovery. Figure 9 shows that if only the register file circuit is modified, recovery is only 7%~37% of the IPC lost¹. Average recovery is less than 20%. This is understandable because physical register file is organized as a circular queue. Faster registers are statically located from entries 0 to 127. Our studies revealed that all committed instructions are evenly distributed in the faster and slower locations of the register file. Depending on the access pattern, the design will be able to recover, at the most, only 50% of the IPC lost. However, the proposed dynamic allocation of registers through duplication increases the chance of using those faster registers. The simulation (Figure 10) shows that, on the average, over 80% of the dynamic instructions take advantage of the un-occupied space and duplicate data into the faster part of the register file, and have shorter register read latency. With dynamic address mapping of the register file, the design enables recovery of more than 50% of the IPC loss, due to extra cycles. Note for both these cases (RF-S-2_3 and RF-D-2_3), data cache latency is 6, while the upper bound used to obtain the IPC loss has a 5-cycle level 1 data cache.

The level 1 data cache also will use the large-signal array design in the near future. The same circuit organization of the register file can be applied to level 1 data cache. However, level 1 cache is not organized in a circular queue and will most likely be occupied in both halves most of the time (except when the line is invalidated). The proposed dynamic addressing scheme cannot be applied as easily. Thus, a static variable latency design can be employed. Both the TAG array and the DATA array of the data cache can take advantage of the physical locality.

During an access, the most significant bit (MSB) of the set address determines whether the data is in the faster (MSB=0) or slower (MSB=1) part of the array. If it is in

¹ Bound=1-(RF-S-3/RF-S-2) and RF-S-2_3-loss=1-(RF-S-2_3)/(RF-S-2) and RF-D-2_3-loss=1-(RF-D-2_3/RF-S-2). Hence, RF-S-2_3-gain=1-(RF-S-2_3/Bound) and RF-D-2_3-gain=1-(RF-D-2_3-loss/Bound)

the faster half, the cache-hit latency is 5 cycles. Otherwise, it is 6-cycle. Figure 11 shows that if this circuit technique is applied to the level 1 data cache with fixed fast and slow locations. This technique recovers 22% of the IPC loss from to super pipelining. When both the register file and data cache employ this proposed method recovers 73% of the IPC loss in average. Different benchmarks will have different effects. In the best case - gcc, the technique is able to recover 96% of the IPC loss.

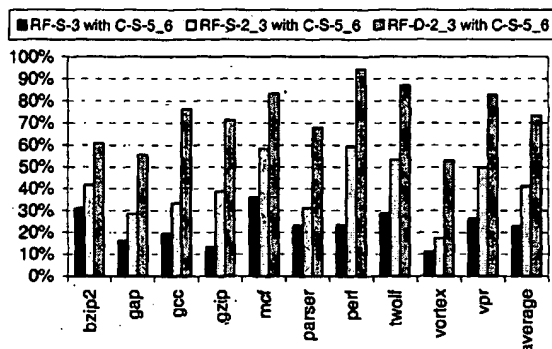


Figure 11. IPC Recovered for Cache Static 5/6 Cycle Reads

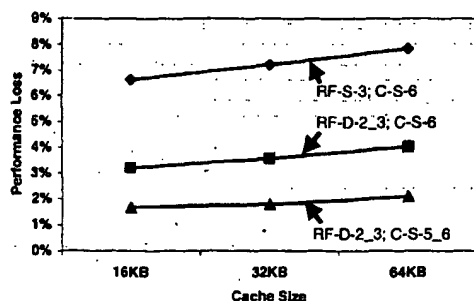


Figure 12. Effect of Cache Size

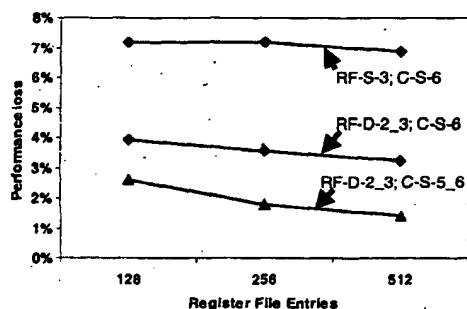


Figure 13. Effect of Register File Size

4.2.4. Sensitivity Due to Cache Size. As level 1 data cache size increases, more memory accesses will hit the cache. As a result, it is observed that the cache-hit latency will affect the IPC degradation more. Figure 12 shows that as level 1 data cache increases to 64KB, IPC loss due to longer latency increases from 7.2% to almost 8%. However, the proposed method used to recover IPC loss is insensitive to level 1 data cache size. The percentage IPC degradation recovered with the proposed method remains approximately the same for three different cache sizes at 75%.

4.2.5. Sensitivity Due to Register File Size. If the number of entries in a register file is small, having longer register file access latency (thus a longer pipeline) tends to apply more pressure on the physical register file. This is because more instructions will be in-flight and a processor will need more physical registers to accommodate them. The average physical register file occupancy rate will be higher and there will be less chance to exploit the faster cycle time access using the proposed re-mapping method. As the physical register file size increases, more un-occupied spaces are available for duplication. Figure 13 shows that the recovery of IPC degradation increases to 80% (an absolute performance loss of only 1.4%) as the size of the register file increases to 512 entries.

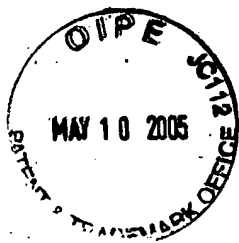
5. Conclusion

As issue-width increases to exploit instruction level parallelism, many memory arrays in the microprocessor must grow in size. Since wiring delay contributes greatly to the access time of an array, multiple cycles are allocated for accessing these arrays. This paper proposes a circuit solution to allow the array to have variable latency, depending on the physical locality to improve the latency of loops with dependent instructions. Furthermore, the paper proposes a method to re-configure the address decoding to take advantage of this asymmetric access time for the physical register file. To validate the new scheme, a microarchitecture simulator is used. With dynamic addressing register file and static variable level 1 data cache, the proposed scheme is able to achieve a recovery of 73% of the IPC lost since pipeline stages are increased. This method is general and can be applied to many of the arrays structures in a modern superscalar microprocessor and shows a good scaling trend as the microprocessor's pipeline is further deepened.

References

- [1] M. Bohr, "Interconnect Scaling - The Real Limiter to High Performance ULSI," 1995 International Electron Devices Meeting, pp. 241-244.

- [2] R. Ho, K. Mai, M. Horowitz, "The Future of Wires," The Proceedings of the IEEE, Vol. 89, No. 4, April 2001.
- [3] V. Agarwal, M.S. Hrishikesh, S. Keckle, D. Burger, "Clock Rate versus IPC: The End of the Road for Conventional Microarchitectures," The Proceedings of the 24th Annual International Symposium on Computer Architecture, 2000, pp. 248-259.
- [4] A. Hartstein, T. Puzak, "The Optimum Pipeline Depth for a Microprocessor," The Proceedings of the 26th Annual International Symposium on Computer Architecture, 2002, pp. 7-13.
- [5] M. Hrishikesh, D. Burger, N. Jouppi, S. Keckler, K. Farkas, P. Shivakumar, "The Optimal Logic Depth Per Pipeline Stage is 6 to 8 FO4 Inverter Delays," The Proceedings of the 26th Annual International Symposium on Computer Architecture, 2002, pp. 14-24.
- [6] E. Sprangle, D. Carmean, "Increasing Processor Performance by Implementing Deeper Pipelines," The Proceedings of the 26th Annual International Symposium on Computer Architecture, 2002, pp. 25-34.
- [7] E. Borch, S. Manne, J. Emer, E. Tune, "Loose Loops Sink Chips," The Proceedings of the 8th Int. Symp. on High Performance Computer Architecture, 2002, pp. 299-310.
- [8] J. Zamalea, J. Llosa, E. Ayguade, M. Valero, "Two-level Hierarchical Register File Organization for VLIW Processors," The proceedings of the 33rd Int. Symp. on Microarchitecture, 2000, pp. 137-146.
- [9] M. Postiff et al., "Integrating Superscalar Processor Components to Implement Register Caching," Int. Conf. On Supercomputing (ICS), 2001, pp. 348-357.
- [10] David Harris, *Skew-Tolerant Circuit Design*, Morgan Kaufmann, May 2000.
- [11] Doug Burger, Todd M. Austin, etc., "SimpleScalar toolset 3.0b," <http://www.simplescalar.com>.
- [12] SPEC 2000 benchmarks suites, <http://www.spec.org>.



Lecture 2: Computer Elements

- Last Time
 - Course Overview
 - Introduction to Computer Architecture
- Today
 - Announcements
 - Reading due: P&H Chapter 1
 - Reading due next class: P&H Chapter 2.1-2.4, Handout #1
 - Computer elements
 - Transistors, wires, pins
 - Introduction to performance

UTCS

Lecture 2

1

Computer Elements

- Transistors (computing)
 - How can they be connected to do something useful?
 - How do we evaluate how fast a logic block is?
- Wires (transporting)
 - What and where are they?
 - How can they be modeled?
- Memories (storing)
 - SRAM vs. DRAM

UTCS

Lecture 2

2

What Comes out of the Fab?

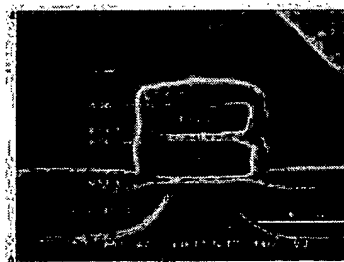


UTCS

Lecture 2

3

The Mighty Transistor!



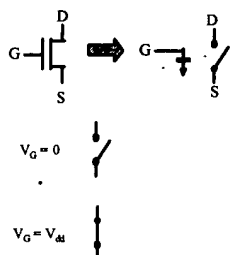
UTCS

Lecture 2

4

Transistor As a Switch

- Ideal Voltage Controlled Switch



UTCS

Lecture 2

5

Abstractions in Logic Design

- In physical world
 - Voltages, Currents
 - Electron flow
- In logical world - abstraction
 - $V < V_{lo} \Rightarrow "0" = \text{FALSE}$
 - $V > V_{hi} \Rightarrow "1" = \text{TRUE}$
 - In between - forbidden
- Simplify design problem

voltage	
V_{dd}	"1"
V_{hi}	"?"
V_{lo}	"0"
0v	

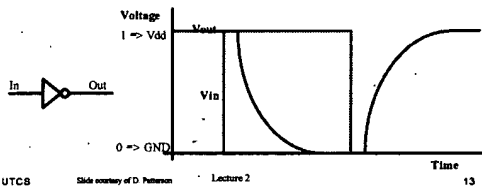
UTCS

Lecture 2

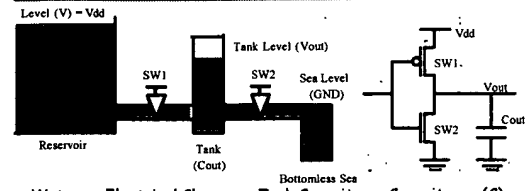
6

Ideal (CS) versus Reality (EE)

- When input 0 \rightarrow 1, output 1 \rightarrow 0 but NOT instantly
 - Output goes 1 \rightarrow 0: output voltage goes from Vdd (5v) to 0v
- When input 1 \rightarrow 0, output 0 \rightarrow 1 but NOT instantly
 - Output goes 0 \rightarrow 1: output voltage goes from 0v to Vdd (5v)
- Voltage does not change instantaneously

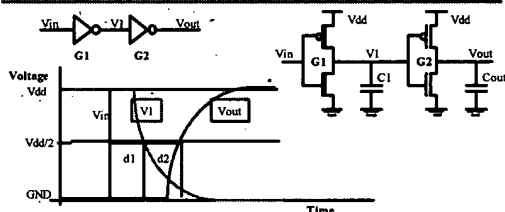


Fluid Timing Model



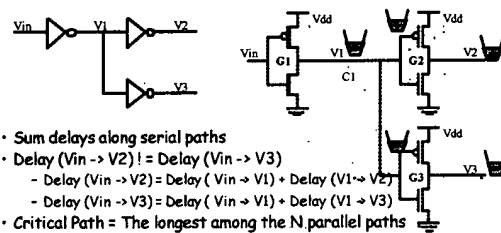
- Water \leftrightarrow Electrical Charge
- Water Level \leftrightarrow Voltage
- Size of Pipes \leftrightarrow Strength of Transistors (G)
- Time to fill up the tank $\sim C/G$
- Tank Capacity \leftrightarrow Capacitance (C)
- Water Flow \leftrightarrow Charge Flowing (Current)
- Resistance $R = 1/G$

Series Connection



- Total Propagation Delay = Sum of individual delays = $d1 + d2$
- Capacitance $C1$ has two components:
 - Capacitance of the wire connecting the two gates
 - Input capacitance of the second inverter

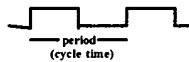
Review: Calculating Delays



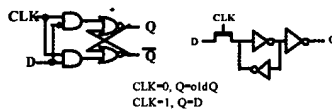
- Sum delays along serial paths
- Delay ($Vin \rightarrow V2$) \neq Delay ($Vin \rightarrow V3$)
 - Delay ($Vin \rightarrow V2$) = Delay ($Vin \rightarrow V1$) + Delay ($V1 \rightarrow V2$)
 - Delay ($Vin \rightarrow V3$) = Delay ($Vin \rightarrow V1$) + Delay ($V1 \rightarrow V3$)
- Critical Path = The longest among the N parallel paths
- $C1$ = Wire C + Cin of Gate 2 + Cin of Gate 3

Clocking and Clocked Elements

- Typical Clock
 - 1Hz = 1 cycle per second



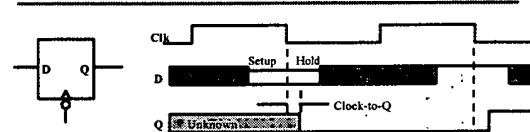
- Transparent Latch



- Edge Triggered Flip-Flop



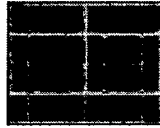
Storage Element's Timing Model



- Setup Time: Input must be stable BEFORE the trigger clock edge
- Hold Time: Input must REMAIN stable after the trigger clock edge
- Clock-to-Q time:
 - Output cannot change instantaneously at the trigger clock edge
 - Similar to delay in logic gates, two components:
 - Internal Clock-to-Q
 - Load dependent Clock-to-Q

Memory

- Moves information in time (wires move it in space)
- Provides state
- Requires energy to change state
 - Feedback circuit - SRAM
 - Capacitors - DRAM
 - Magnetic media - disk
- Required for memories
 - Storage medium
 - Write mechanism
 - Read mechanism



4Gb DRAM Die

UTCS

Lecture 2

25

Technology Scaling Trends

- CPU Transistor density - 60% per year
- CPU Transistor speed - 15% per year
- DRAM density - 60% per year
- DRAM speed - 3% per year
- On-chip wire speed - decreasing relative to transistors (witness the Pentium 4 pipeline)
- Off-chip pin bandwidth - increasing, but slowly
- Power - approaching costs limits
 - $P = CV^2f + I_{leak}V$
- All of these factors affect the end system architecture

UTCS

Lecture 2

26

Summary

- Logic Transistors + Wires + Storage = Computer!
- Transistors
 - Composable switches
 - Electrical considerations
 - Delay from parasitic capacitors and resistors
 - Power ($P = CV^2f$)
- Wires
 - Becoming more important from delay and BW perspective
- Memories
 - Density, Access time, Persistence, BW

UTCS

Lecture 2

27

Next Time

- Evaluation of Systems
 - Basic instruction sets, arithmetic operations
 - Computer arithmetic (integer)
- Reading assignment
 - P&H Chapter 2.1-.24, Handout #1
- HW #1 - Due Jan. 27 in class

UTCS

Lecture 2

28

1. Designing a 64x32-bit memory array – Background

Memory arrays are an essential building block of all digital systems. In this semester's project, we will design a memory array that consists of 64 32-bit words.

1.1. High level structure

A block diagram of a memory array is shown in Figure 1.

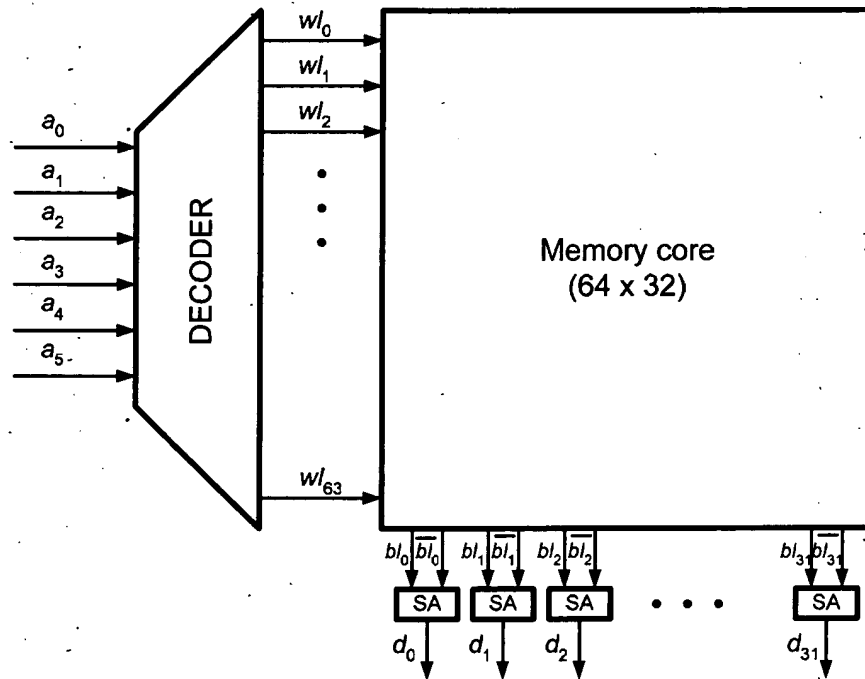


Figure 1. SRAM array block diagram.

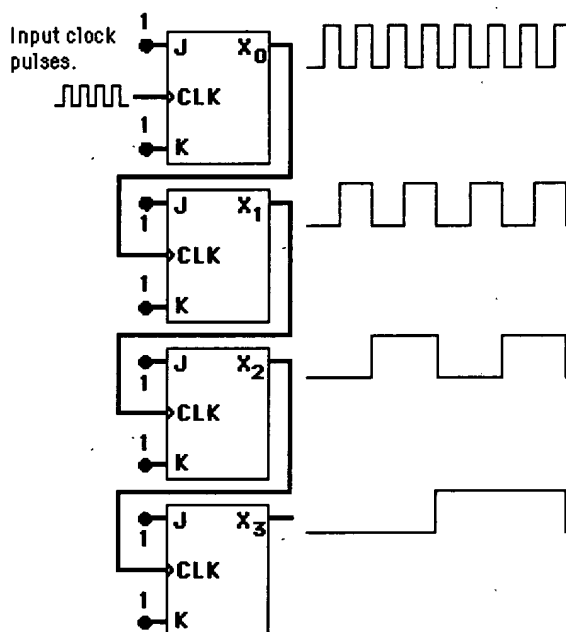
There are three major parts:

- Address decoder: The address decoder takes in 6 address lines, provided as true and complement values a_5 - a_0 and $\overline{a_5}$ - $\overline{a_0}$ and produces 64 select signals wl_{63} - wl_0 .
- SRAM array: Consists of an array of 64 x 32 6-transistor SRAM cells.
- Sense amplifiers: There is one sense amplifiers per column that amplifies the SRAM cell outputs. There are no column decoders – 32-bit words are addressed.

In addition to these blocks the array also has a circuitry that allows writing the data and precharging the bitlines to V_{DD} before the read operation, which is not shown in figure.

Binary Counting

A binary counter can be constructed from J-K flip-flops by taking the output of one cell to the clock input of the next. The J and K inputs of each flip-flop are set to 1 to produce a toggle at each cycle of the clock input. For each two toggles of the first cell, a toggle is produced in the second cell, and so on down to the fourth cell. This produces a binary number equal to the number of cycles of the input clock signal. This device is sometimes called a "ripple through" counter. The same device is useful as a frequency divider.



Decade Counter

Flip-Flops

Index

Electronics concepts

Digital circuits

Data Transfer

J-K Flip-Flop Applications

Reference Tocci Digital Systems, Sec 5-19

HyperPhysics*****Electricity and magnetism

R
Nave

Go Back

**This Page is Inserted by IFW Indexing and Scanning
Operations and is not part of the Official Record**

BEST AVAILABLE IMAGES

Defective images within this document are accurate representations of the original documents submitted by the applicant.

Defects in the images include but are not limited to the items checked:

- ☐ BLACK BORDERS
- ☐ IMAGE CUT OFF AT TOP, BOTTOM OR SIDES
- ☒ FADED TEXT OR DRAWING
- ☐ BLURRED OR ILLEGIBLE TEXT OR DRAWING
- ☐ SKEWED/SLANTED IMAGES
- ☐ COLOR OR BLACK AND WHITE PHOTOGRAPHS
- ☐ GRAY SCALE DOCUMENTS
- ☒ LINES OR MARKS ON ORIGINAL DOCUMENT
- ☐ REFERENCE(S) OR EXHIBIT(S) SUBMITTED ARE POOR QUALITY
- ☐ OTHER: _____

IMAGES ARE BEST AVAILABLE COPY.

As rescanning these documents will not correct the image problems checked, please do not report these problems to the IFW Image Problem Mailbox.